Linking

Simple translation will work

m.c  ASCII source file

translator

→ binary executable file
  (memory image on disk)
problems

- efficiency  small change to source req. complete decompilation
- modularity  hard to share common function (e.g. printf)

solution

linker (static linker)
m.c  
    ↓
    translate
    ↓
    m.o
    ↓
    link (ld)
    ↓
    p

a.c  
    ↓
    translate
    ↓
    a.o
    ↓
    object file
    relocatable
    
executable object file
resolves external references

symbols declared in file, used elsewhere

relocate symbols: from relative дек in .o

file to new absolute positions

update all references to relocated symbols

can be code:  all //symbol

Data:  int * xp = fX;  //symbol x
Executable and Linkable Format (ELF)

- std. format for many Unix-style os
- relocatable obj files
- executable binaries
- so shared object files
Contents of ELF file:

**ELF header**
- Magic number, file type, machine
- Program header table entry (segments)

**Program header table**
- Section header table entry (sections)
- Text section
- Data section
- BSS section
- Symbol section
- Rel. .txt
- Rel. .data
- Debug
- Line
- Str. Tab
- Section header table

**Section header table**
- Location of sections within ELF
- Initialized data
- Uninitialized data
- Global symbols
- Relocation info for code
- Data
- Debugging symbol table (-g)
- Maps line #s to machine code
- Constant strings table
```c
int e = 7;
int main() {
    int r = a();
    int x = 15;
    int y;
}
int a() {
    return (e + x + y);
}
```
program symbols are "strong" or "weak"
- **strong**: procedure names and initialized global variables
- **weak**: uninitialized global variables

```
pl.c

int foo = 5;

pl()} {

plonly

3

p2.c

int foo;

p2()} {

3

weak

strong
```
Linking Rules

1. A strong symbol can appear only once.

2. A weak symbol can be overridden by a strong symbol of the same name.
   - `refs` to weak symbol resolve to strong sym.

3. If there are multiple weak symbols, the linker can pick arbitrary one.
\[\text{int } x\]
\[p1(1), \{3\}\]
\[\text{int } x\]
\[p1(1), \{3\}\]
\[\text{error!}\]

\[\text{int } x\]
\[p1(y), \{3\}\]
\[\text{double } x\]
\[p2(w), \{3\}\]
\[\text{Nasty}\]

\[\text{int } x\]
\[p1(y), \{3\}\]
\[\text{double } x\]
\[p2(w), \{3\}\]
\[\text{Error!}\]
`pl.c` -> `compil` -> `pl.o`

`p2.c` -> `compil` -> `p2.o`

`linker` -> `p` (executable object file)

includes only functions in `libc` that are ref. by `pl`, `p2`
```
m.c \downarrow
m.o
\downarrow
```

```
a.c \downarrow
a.o
```

```
partially
linked
exe.c
```

```
\text{loader/dynamic linker}/ \text{ld-linux.so}
```

```
\text{shared library}
```

```
\text{fully linked/}
\text{executed}
```

```
p
```

```
p
```